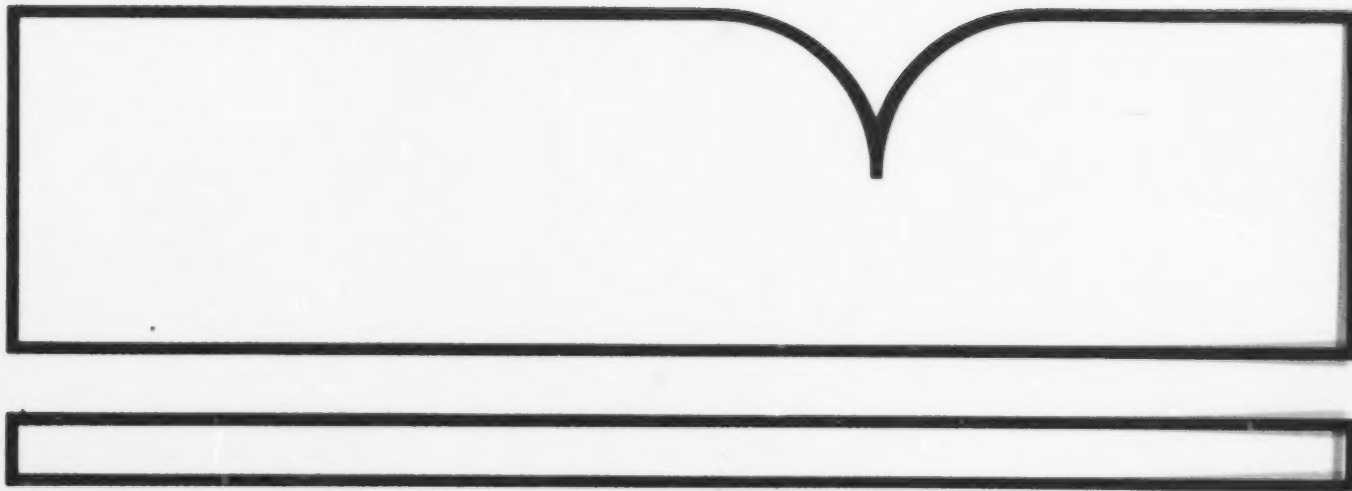


PB86-21 21 31

Portability, Maintenance and
Fortran Programming Style

(U.S.) Environmental Research Lab., Athens, GA

Jun 86



U.S. Department of Commerce
National Technical Information Service

NTIS

EPA/600/D-86/123
June 1986

PORTABILITY, MAINTENANCE AND FORTRAN PROGRAMING STYLE

by

Thomas O. Barnwell, Jr., and David W. Disney*

Assessment Branch
Environmental Research Laboratory
Athens, GA 30613

*Computer Sciences Corporation
Environmental Research Laboratory
Athens, GA 30613

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
ATHENS, GA 30613

TECHNICAL REPORT DATA		
(Please read Instructions on the reverse before completing)		
1. REPORT NO. EPA/600/D-86/123	2.	3. RECIPIENT'S ACCESSION NO. PB86 212131/AS
4. TITLE AND SUBTITLE PORTABILITY, MAINTENANCE AND FORTRAN PROGRAMING STYLE	5. REPORT DATE June 1986	
	6. PERFORMING ORGANIZATION CODE	
7. AUTHOR(S) Thomas O. Barnwell, Jr., and David W. Disney*	8. PERFORMING ORGANIZATION REPORT NO.	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Environmental Research Laboratory, U.S. Environmental Protection Agency, Athens GA 30613 *Computer Sciences Corporation, U.S. Environmental Protection Agency, Athens GA 30613	10. PROGRAM ELEMENT NO. ANWB1E	
	11. CONTRACT/GRANT NO.	
12. SPONSORING AGENCY NAME AND ADDRESS Environmental Research Laboratory - Athens, GA Office of Research and Development U.S. Environmental Protection Agency Athens, GA 30613	13. TYPE OF REPORT AND PERIOD COVERED	
	14. SPONSORING AGENCY CODE EPA/600/01	
15. SUPPLEMENTARY NOTES		
16. ABSTRACT Programing conventions are described that make FORTRAN programs easily transportable from minicomputers and mainframes to the microcomputer environment. Topics discussed include program design considerations, user interface, documentation, coding conventions, downloading problems, and development tools.		
17. KEY WORDS AND DOCUMENT ANALYSIS		
a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group
18. DISTRIBUTION STATEMENT RELEASE TO PUBLIC	19. SECURITY CLASS (This Report) UNCLASSIFIED	21. NO. OF PAGES 11
	20. SECURITY CLASS (This page) UNCLASSIFIED	22. PRICE

NOTICE

This document has been reviewed in accordance with U.S. Environmental Protection Agency policy and approved for publication. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

Portability, Maintenance and FORTRAN Programing Style

Thomas O. Barnwell, Jr.¹
David W. Disney²

Introduction

At the Center for Water Quality Modeling, we have evolved a set of programing conventions that make our FORTRAN programs easily maintainable and transportable. These conventions have helped us in transporting many of our supported programs from minicomputers and mainframes to the microcomputer environment. This paper describes these programing conventions and development tools and discusses how they have aided in moving our programs to microcomputers while still providing a workable environment for the maintenance of code on different hardware and software systems.

Program Design Considerations

It is appropriate to discuss two aspects of programing style -- design considerations and coding conventions. Design relates to the overall structure of the program while coding conventions refer to the the actual FORTRAN implementation. Much has been written about structured programing and software design and we have found this design concept to be useful. The following paragraphs outline our approach to structured programing in FORTRAN.

Possibly the most important design consideration is modularity of structure. Modularity is fostered by writing small subroutines. To borrow a rule from the writing profession, be brief and concise. Treat a subroutine as you would a paragraph and confine it to a single thought -- a subroutine should only do one thing. A good rule is to restrict a subroutine to two pages or less; some prefer to limit a subroutine to a single page. But a single page may be too restrictive if your style uses a lot of open space to increase understanding. Always keep in mind that both you and others will have to read (and maintain) what you write.

-
1. Center for Water Quality Modeling
 2. Computer Sciences Corp.
Environmental Research Laboratory
U S EPA, Athens, Ga 30613

COMMON Blocks are a major part of our design. Although FORTRAN does not have code structures like many other languages, code structures such as a common work or data area can be emulated with COMMON Blocks. We also prefer that data be passed between subroutines using COMMON Blocks rather than Arguments, particularly at higher levels of the program. Although there are some arguable advantages to using arguments for this purpose (after all, that is their sole *raison d'être*), they can be troublesome. As discussed later, we have found them to be a frequent source of problems.

Of course, modular structure is much more than writing small subroutines and using COMMON Blocks. A clear specification of objectives and plan for achieving them is at the heart of structured programming. But these principles are a good beginning.

Although there may be good reasons for not considering it, portability is another important design consideration. Our programs are written to be used by others, mostly on their own computers and the programs have a surprisingly long lifetime. Although we cannot claim credit for the original design, QUAL2E (Brown and Barnwell, 1985) is based on a structure set out over 15 years ago. We use only 1977 ANSI Standard FORTRAN and avoid extensions to the standard.

If writing single-purpose code for a specific application, the programmer is justified in using the FORTRAN extensions available on his computer. In general, these extensions are provided to enhance the efficiency of both the programmer and the program. But if there is the slightest inkling that the program will be used on different hardware, the portability issue must be considered.

To insure portability, it is useful to identify target compilers that represent a reasonable spectrum of computers. For example, U S EPA staff can count on the availability of at least three kinds of computer -- IBM PC compatible, DEC VAX technology, and IBM 30xx-series mainframes. We run our code through compilers on each of these systems.

Where hardware-specific code must be used, the programmer must isolate and clearly identify this dependent code. For example, DATE and TIME subroutine calls are a common extension that are not part of the ANSI-77 standard but are available on many systems. Although not hardware-specific, file opening and closing conventions may vary among installations, even for the same type of hardware, and it is good practice to isolate these statements in a separate, clearly identified subroutine.

User Interface

The user interface is becoming more and more important to the computer programmer. The "friendliness" of programs in the microcomputer environment such as LOTUS 123™ have raised user expectations so high that the traditional batch-oriented card file input is fast becoming unacceptable. In fact, the Center for Water Quality Modeling has begun a major effort to develop a consistent interactive user interface for all our programs.

In designing a user interface, the programmer should strive to use complete English sentences for such things as error messages, status diagnostics, or run-time prompts. Although inside jargon and computerese may be expedient in the short run, the novice user will often be confused by such cryptic statements as "PROGRAM STOP - INPUT ERROR." Even an experienced programmer may wonder if this message was generated by the operating system or the program itself.

An important function of the user interface is verification of all inputs. Although the user interface may not be able to catch subtle errors in program input, several standard checks should be made. Perhaps the most obvious is checking of character versus numeric input. How many of us have not spent considerable time tracking down an error to find that its cause is a letter "O" where the number "0" should be, or vice-versa?

Another good practice is to explicitly check for real and integer numbers and to check that numbers are in a valid range. Also, default values should be stated or displayed explicitly. Cases exist where widely used computer programs with perhaps inappropriate default parameter values have inadvertently established "standard practice."

It is useful to place input verification criteria and defaults in tables or external files. This makes the criteria easy to locate and change. The use of an external file is particularly convenient as changes can be made without recompiling the program.

An echo of inputs should be provided as an option so that the user can document input when desired but switch it off when it is not needed. As most programs have the option of running either interactively or batch, the programmer should provide program traps to stop execution when input errors are uncovered rather than let execution continue.

And when writing tabular output from a program, concise but descriptive headings should be provided and output formatted for easy reading. Nothing is more frustrating than 10 pages of densely packed numbers with no identification. Include your documentation on the output, not in the users manual.

Documentation

Documentation is critical to a successful computer program. We think of documentation targeted both for the user and for the programmer. User documentation should include a description of the application or use of the computer program. The exact equations and solution techniques used as well as guidance for the user on ranges and typical values of input variables should be provided.

The actual installation of the program on the users' computer system, however, is often done by a programmer who is interested in quite different things. The programmer is interested in what we call "implementation documentation" that contains detailed instructions on how to install and maintain the program on a user's system.

Both the programmer and the user are interested in sample input and output to verify that the installation is correct and the program is operating correctly on their system. The system or implementation documentation also should include a functional organization of source code files by name as well as descriptions of program variables (particularly global variables) in case the user and his programmer have to troubleshoot the software. Often, the user may want to read the program in order to better understand what it is doing. This should be made as easy as possible by making available copies of the source code and identifying development and maintenance software.

Coding Conventions

Comments are an essential part of the source code and are often the only source code documentation. Although pseudocode is useful when designing a large program, we have found it difficult and costly to keep current. Each subroutine or program segment should have introductory comments that clearly state its purpose; describe inputs, outputs and modified variables; and identify error conditions. Comments are useful to physically separate sections of code that do not justify isolation in a subroutine and to explain loops and conditional statements. These are often the hardest parts of a FORTRAN program to decipher; particularly if the dreaded GO TO is used with indiscretion. (We all know the rule on GO TO -- DON'T!!!)

Complete English statements should be used; many programmers use a shorthand that can be completely indecipherable. Others will have to read what you write and base the maintenance and understanding of the source code on your commentary. When trying to decipher a program, remember that comments are not infallible -- they can lie.

Just as indentation and physical separation are used to help delineate parts of a composition, these devices should be used to graphically explain program logical flow and subordination. Loops should be indented and should end in the same column they begin. IF statements and IF-THEN-ELSE structures also should follow a consistent indentation format. Comments may be helpful in separating parts of loops and conditional statements.

Indentation is useful when it is necessary to continue COMMON, DATA and other type and declaration statements beyond a single line. For example, beginning the continuation lines at, say, column 10 helps to delineate blocks of code. And if there is no alternative to using a GO TO construct (remember the old maxim -- rules are for privates and lieutenants; experienced programmers know when to break the rules), indentation is useful to help identify the continuations.

Modularity and structure in code can also be enhanced by placing COMMON Blocks in separate source code files that may be brought into other source code files via FORTRAN's INCLUDE statement. In addition to producing smaller subroutines, use of INCLUDE files for COMMON blocks will insure that COMMONs are consistent across a program. Don't dimension or name variables differently in different subroutines. If it is necessary to rename variables, use the EQUIVALENCE statement.

It also is useful to include data type specifications in INCLUDE files. If possible, group different type variables (eg, CHARACTER, INTEGER and REAL) into separate COMMON blocks. This practice will help avoid boundary alignment problems and enhances the maintainability of a program. Explicit declarations of INTEGER*2 and INTEGER*4 data type are also helpful. Most computers default to the INTEGER*4 data type but, on microprocessors in particular, the INTEGER*2 calculations are considerably faster than INTEGER*4 arithmetic (Microsoft, 1985) in addition to saving some memory.

We have found it desirable to isolate file OPEN and CLOSE statements in a central, clearly identified routine because, as mentioned above, these statements may be installation-dependent. It also is good practice to use variables for unit numbers in input-output statements and assign these variables in an "environment" INCLUDE file, or within in the file-open subroutine. Always open terminal input-output units explicitly. Although some installations assume units 5 and 6 for terminal input and output, respectively, others will use unit 5 for both and still others may use units 1 and 2.

Where possible, the user should be allowed to specify file names either at execution or at run time rather than specify the names in the compiled code. This will allow easier installation of code. In addition, the OPEN statement error return codes should be used to recover open errors gracefully, giving explicit error messages using complete English statements.

We favor the use of the PARAMETER statement or a BLOCK DATA subroutine to declare array dimensions for system-wide variables, DO loops, I/O units and other appropriate system wide parameters. Proper use of the PARAMETER statement will allow easy modification of program limits by modifying a single subroutine or source code file brought into the program with an INCLUDE file. This feature is useful if one wishes to modify a program for different machine memory configurations. For example, the IBM/PC version of the QUAL2E program is distributed with array dimensions set for a 256K machine. These array dimensions can be changed by modifying one line in a single subroutine. One should be careful, however, not to use PARAMETER to set values for variables. Remember that the program source code must be edited, recompiled, and linked, to change any PARAMETER or a variable initialized in a BLOCK DATA subroutine.

In defining variables, explicit declarations of variable type should be used rather than the default type. A useful extension on some compilers is the IMPLICIT NONE statement. This statement will force explicit declaration of all variables when writing code and can be easily removed for the distribution version. Dimensioning variables in INTEGER and REAL Statements was considered good practice but we have found this to be a problem with microcomputer compilers that only allow dimensioning in the DIMENSION and COMMON statements and require type specification in separate statements. As mentioned above, the same name should be used everywhere. This helps avoid later confusion when debugging the code. Prefix/Suffix-Type variable names are also useful in helping readability. And a clear specification of initial value avoids problems with systems where a zero default initial value is not the convention.

Downloading Problems

In the process of downloading our software to the microcomputer, we have encountered several problems that relate to programing style and coding conventions discussed above. In general, microcomputer FORTRAN compilers are quite strict in following the ANSI standard for FORTRAN 77. Few if any FORTRAN language extensions are available on microcomputers and practices that are acceptable on larger computers are not acceptable with these compilers.

For example, the compilers are quite sensitive to statement order and will not allow mixing of specification, declaration and executable statements. Variables that are assigned character string values should be defined as Character in a type specification. Programmers should be careful not to use variable names longer than six characters as the more restrictive compilers will not recognize differences beyond the sixth character.

A good practice is to separate character and numeric data into individual COMMON blocks to avoid problems with boundary alignment. The same is true of integer, real and double precision variables. And one should avoid mixing these different type variables in executable statements, always using explicit type conversions in arithmetic statements.

Another good practice is to initialize variables in a BLOCK DATA subroutine and avoid reinitializing with DATA statements. On micros, argument types must be compatible in CALL and SUBROUTINE statements. Although compilers on larger computers may convert between types, the microcomputer compilers do not.

Some compilers will allow FORMAT statement field separators to be omitted -- on micros, this will cause errors. ENCODE and DECODE statements should not be used; they have been replaced by internal READ and WRITE statements in FORTRAN 77.

Development Tools

There are a number of FORTRAN compilers available for personal computers. Two recent reviews in PC Tech Journal (Howard, 1985) and Computer Language (Benson et al., 1986) discussed nine different FORTRAN compilers (Table 1) for the MS-DOS operating system and the Macintosh.

Several letters responding to the Howard's review are contained in the March 1986 issue and provide additional insight to the compilers. Of particular interest is an editorial in the March 1986 PC Tech Journal (Fastie, 1986) that observes "the volume of comment on our coverage indicates considerably larger interest in FORTRAN than expected ... confirming that FORTRAN is alive and well. ... One particularly surprising comment is about FORTRAN's portability. Because it is not a systems programmers language, programmers are not as likely to exploit extensions supplied by a particular vendor, opting instead for a more standard approach. This portability extends beyond the desktop to the minicomputers and mainframes crunching FORTRAN."

TABLE 1. REVIEWS OF PC FORTRAN COMPILERS

Compiler	PC Tech Journal Oct. 85	Computer Language Jan. 86
IBM Professional FORTRAN (a.k.a. Ryan-McFarland)	X	X
Lahey	X	X
Microsoft	X	X
Digital Research	X	X
Intel FORTRAN-86		X
Prospero ProFORTRAN		X
Supersoft		X
WATFOR-77		X
Microsoft MACFORTRAN		X

The Computer Language article is particularly interesting because it also discusses a number of add-on tools that provide graphics, string manipulation, windows, scientific subroutines, and productivity enhancements.

Our experience is limited to the Microsoft™ and IBM Professional™ (a.k.a. Ryan-McFarland™) FORTRAN (ProFORT) compilers. Microsoft's compiler is flexible in that it can produce executable code that executes on PCs without the 8087 Math Coprocessor chip, whereas ProFORT requires the coprocessor. However, Microsoft's FORTRAN syntax, particularly for INCLUDE statements, and use of Metacommands for compiler options differ from conventional usage.

We have found that ProFORT is compatible with our primary development environment, DEC VAX FORTRAN. ProFORT is a full implementation of FORTRAN 77 with few, if any, extensions and is often 100% compatible with other FORTRAN development environments. This compatibility has been a key to our ability to develop and maintain software targeted for a wide variety of computer systems.

The key to installing large FORTRAN programs on the PC lies in the linkage editor. The LINK command provided with both the Microsoft and ProFORT compilers will allow only one level of overlay and will only allow overlay of code segments. In trying to link a large program (25,000 lines of code) with these linkage editors, we received a message indicating that code segment DGROUP exceeded 64K (Parker, 1986, discusses this problem). Phoenix Software's PLINK86 linkage editor allowed us to bypass this problem by allowing multiple overlay levels (up to 32 levels of overlay may be used). A complex overlay structure may be required by large program size and proper software design can make for a much more efficient overlay structure.

Summary

We have reviewed programing design and coding conventions that have served us well in maintaining and transporting out FORTRAN programs among a variety of computers. The development tools discussed above make it possible to install quite large FORTRAN programs on microcomputers and our experience is showing that the microcomputer can be an attractive environment for running these programs. When execution times are excessive, the programs can be easily moved to larger machines to take advantage of their faster speed.

Disclaimer

Mention of trade names or commercial products does not constitute endorsement or recognition for use by the U. S. Environmental Protection Agency.

References

Bensor, R. et al., "Software Review: FORTRAN on the MICRO", Computer Language, Computer Language Publishing Ltd., San Francisco, CA, Jan 86, pp83-110.

Brown L. C. and T. O. Barnwell, Computer Program Documentation for the Enhanced Stream Water Quality Model QUAL2E, EPA/600/3-85/065, U S EPA, Athens, GA, 30613, Aug 1985.

Fastie, W., "Language Surprises," PC Tech Journal, Ziff-Davis, NY, Mar 1986, p12.

Howard, A., "FORTRAN Options," PC Tech Journal, Ziff-Davis, NY, Oct 1985, pp149-160.

Parker, P., in "Letters", PC Tech Journal, Ziff-Davis, NY, Mar 1986, p22.

_____, IBM Personal Computer Professional FORTRAN, by Ryan-McFarland Corp., IBM Corp., NY 1985.

_____, Microsoft FORTRAN Reference Manual, Microsoft Corp., Bellevue, WA., 98009